



Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Комп’ютерний практикум №2

Технології паралельних обчислень

Тема: Розробка паралельних алгоритмів множення матриць та дослідження їх ефективності

Виконав

студент групи ІІІ-11:

Панченко С. В.

Перевірила:

Стеценко І.В.

Київ 2024

ЗМІСТ

1 Завдання.....	6
2 Виконання.....	7
2.1 Алгоритми.....	7
2.2 Тестовий клас алгоритмів.....	8
2.3 Математична бібліотека.....	11
3 Висновок.....	13
4 Посилання.....	14
ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ.....	15

1 ЗАВДАННЯ

1. Реалізуйте стрічковий алгоритм множення матриць. Результат множення записуйте в об'єкт класу Result. 30 балів.
2. Реалізуйте алгоритм Фокса множення матриць. 30 балів.
3. Виконайте експерименти, варіюючи розмірність матриць, які перемножуються, для обох алгоритмів, та реєструючи час виконання алгоритму. Порівняйте результати дослідження ефективності обох алгоритмів. 20 балів.
4. Виконайте експерименти, варіюючи кількість потоків, що використовується для паралельного множення матриць, та реєструючи час виконання. Порівняйте результати дослідження ефективності обох алгоритмів. 20 балів.

2 ВИКОНАННЯ

2.1 Алгоритми

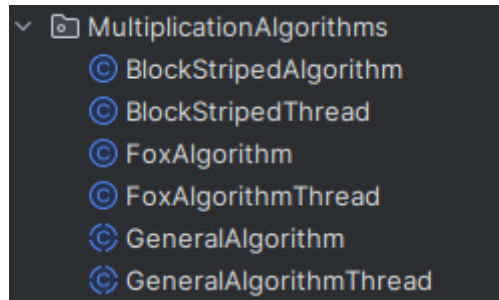


Рисунок 2.1.1 - Структура проекту

Розглянемо структуру:

- 1) `GeneralAlgorithm` — абстрактний клас, який узагальнює використання алгоритму множення матриць.
- 2) `GeneralAlgorithmThread` — абстрактний клас, який узагальнює використання потоків паралельними алгоритмами множення матриць.
- 3) `BlockStripedAlgorithm` та `FoxAlgorithm` — дочірні класи `GeneralAlgorithm` паралельного множення матриць, у конструктори яких передається кількість потоків та дві матриці.
- 4) `BlockStripedAlgorithmThread` та `FoxAlgorithmThread` — дочірні класи `GeneralAlgorithmThread`, що надають можливість розділити множення матриць на потоки.

Покажемо перемноження матриць на прикладі `main`-функції класу `FoxAlgorithm`, де виконується множення матриць за допомогою `FoxAlgorithm` та `BlockStripedAlgorithm`.

```

1 public FoxAlgorithm() {}
2
3 > public FoxAlgorithm(int threadsNum, Matrix2D first, Matrix2D second) { super(thr
4
5
6
7 public static void main(String[] args) {
8     var matrixFactory = new Matrix2DFactory();
9     var rows = 3;
10    var cols = 3;
11    var minVal = 0;
12    var maxVal = 10;
13    var threadsNum = 3;
14
15    var first : Matrix2D = matrixFactory.getRandom(rows, cols, minVal, maxVal);
16    var second : Matrix2D = matrixFactory.getRandom(rows, cols, minVal, maxVal);
17
18    var algorithm = new FoxAlgorithm(threadsNum, first, second);
19    var striped = new BlockStripedAlgorithm(threadsNum, first, second);
20    var result : Matrix2D = algorithm.solve();
21    var stripedResult : Matrix2D = striped.solve();
22    System.out.println("First:\t" + first);
23    System.out.println("Second:\t" + second);
24    System.out.println("Fox:\t" + result);
25    System.out.println("Striped:\t" + stripedResult);

```

Рисунок 2.1.2 - Main-функція класу FoxAlgorithm

Покажемо результати виконання функції.

```

First:  {{1.0, 7.0, 0.0}, {6.0, 7.0, 7.0}, {1.0, 7.0, 3.0}}
Second: {{9.0, 7.0, 8.0}, {0.0, 6.0, 6.0}, {4.0, 3.0, 0.0}}
Fox:    {{9.0, 49.0, 50.0}, {82.0, 105.0, 90.0}, {21.0, 58.0, 50.0}}
Striped: {{9.0, 49.0, 50.0}, {82.0, 105.0, 90.0}, {21.0, 58.0, 50.0}}

```

Рисунок 2.1.3 - Результат множення

2.2 Тестовий клас алгоритмів

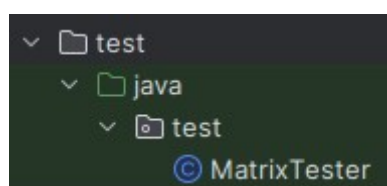


Рисунок 2.2.1 - Структура проєкту

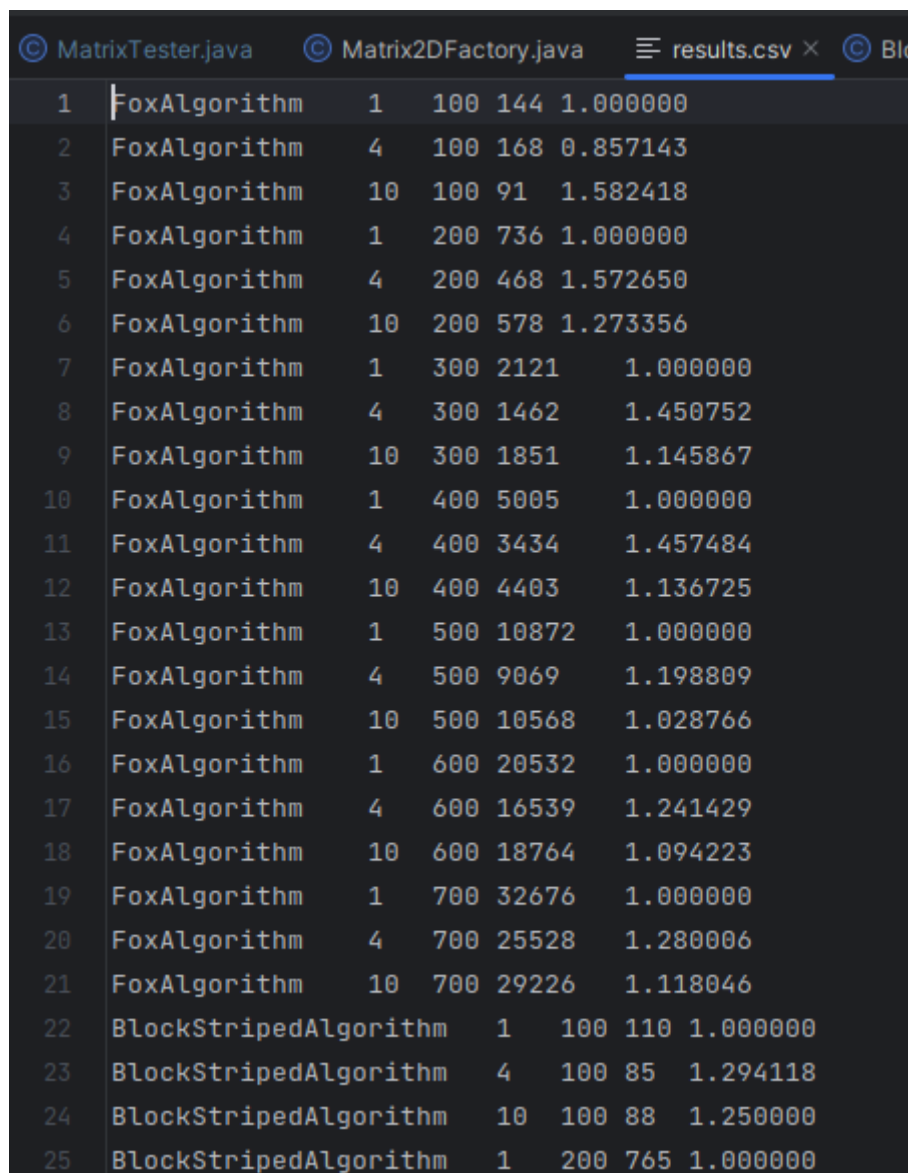
Розглянемо структуру:

- 1) MatrixTester — клас, що проводить тестування результатів множення

матриць, варіюючи кількість потоків, розміри матриць, алгоритми та⁹ малюючи графік.

Продемонструємо різницю роботи алгоритмів та часу виконання алгоритмів.

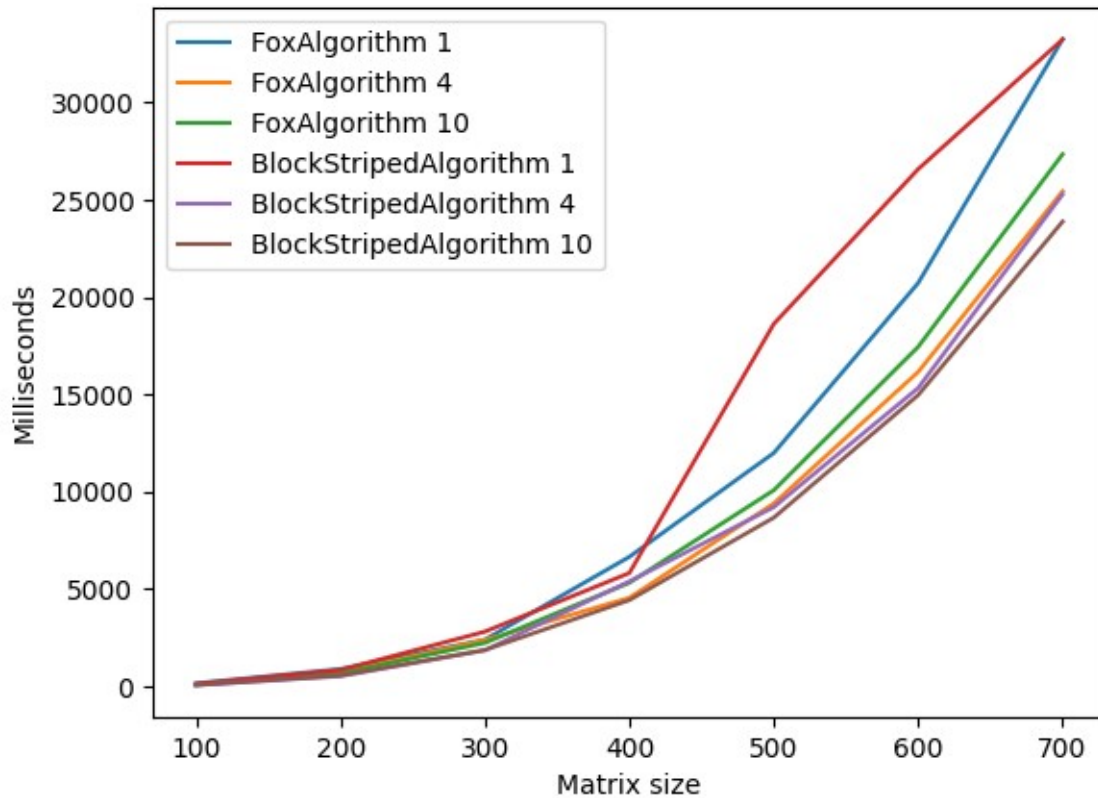
Під час виконання результати записуються у файл results.csv. Покажемо вміст цього файлу. У першій колонці присутня назва алгоритму, у другій — кількість потоків, у третій — розмір матриці, у четвертій — час в мілісекундах, у п'ятій — прискорення у порівнянні з одним потоком.



1	FoxAlgorithm	1	100	144	1.000000
2	FoxAlgorithm	4	100	168	0.857143
3	FoxAlgorithm	10	100	91	1.582418
4	FoxAlgorithm	1	200	736	1.000000
5	FoxAlgorithm	4	200	468	1.572650
6	FoxAlgorithm	10	200	578	1.273356
7	FoxAlgorithm	1	300	2121	1.000000
8	FoxAlgorithm	4	300	1462	1.450752
9	FoxAlgorithm	10	300	1851	1.145867
10	FoxAlgorithm	1	400	5005	1.000000
11	FoxAlgorithm	4	400	3434	1.457484
12	FoxAlgorithm	10	400	4403	1.136725
13	FoxAlgorithm	1	500	10872	1.000000
14	FoxAlgorithm	4	500	9069	1.198809
15	FoxAlgorithm	10	500	10568	1.028766
16	FoxAlgorithm	1	600	20532	1.000000
17	FoxAlgorithm	4	600	16539	1.241429
18	FoxAlgorithm	10	600	18764	1.094223
19	FoxAlgorithm	1	700	32676	1.000000
20	FoxAlgorithm	4	700	25528	1.280006
21	FoxAlgorithm	10	700	29226	1.118046
22	BlockStripedAlgorithm	1	100	110	1.000000
23	BlockStripedAlgorithm	4	100	85	1.294118
24	BlockStripedAlgorithm	10	100	88	1.250000
25	BlockStripedAlgorithm	1	200	765	1.000000

Рисунок 2.2.2 - Результати алгоритму

Не досить зрозуміло, тому у додачу до файлу будувється графік та зберігається у файл “plot.png”.

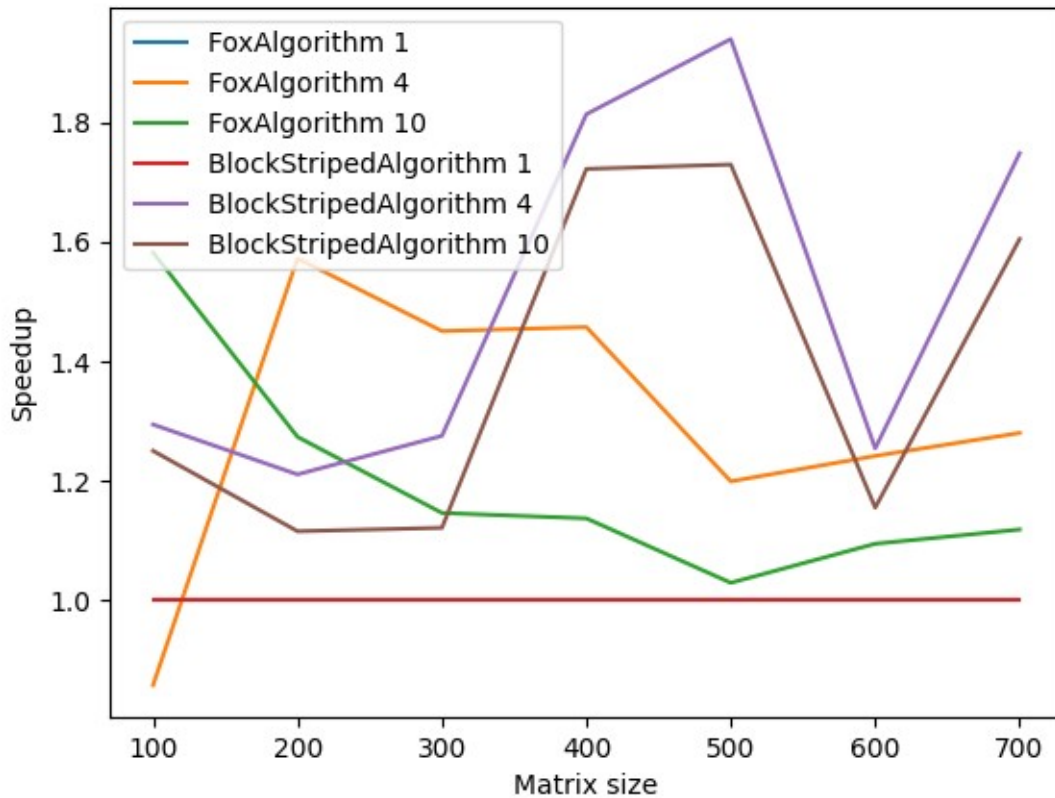


У легенді позначено алгоритми та кількість потоків, що використовувалася.

Бачимо, що зі збільшенням розмірності матриць для усіх алгоритмів збільшується час їхнього виконання, а особливо це помітно для стрічкового алгоритму, що працює на одному потоці.

Якщо ж збільшувати кількість потоків, то бачимо що час виконання зменшується. Однак можна помітити, що збільшення потоків все менше впливає на прискорення алгоритму, що відповідає відповідно закону Амдала[1].

Покажемо на прикладі прискорення, що зазнали алгоритми.



На прикладі FoxAlgorithm4 та BlockStriped4 бачимо, що вплив прискорення спадає при збільшенні розмірності матриць.

2.3 Математична бібліотека

Попередньо з першою лабораторною роботою дописав функціонал математичної бібліотеки, розділивши її на інтерфейси, перерахунки, матриці, вектори.

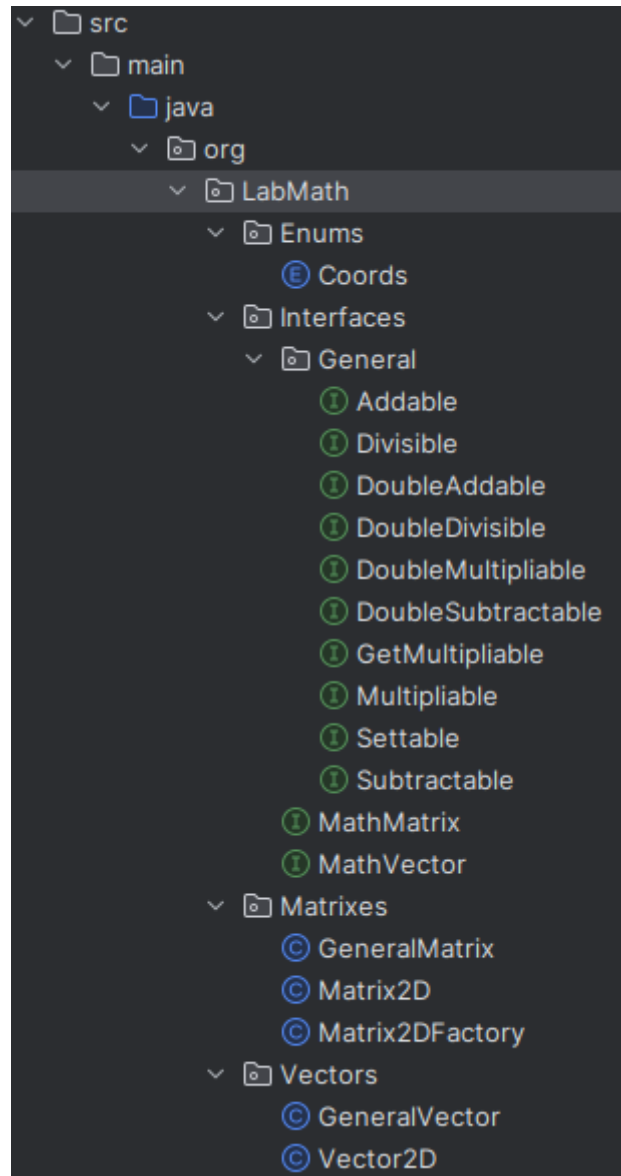


Рисунок 2.3.1 - Структура проєкту.

Розглянемо структуру:

- 1) Enums — пакет, що вміщує всі enum`и, що використовуються(Coords — enum координат);
- 2) Interfaces — пакет, що вміщує прості інтерфейси General, які описують певну просту властивість класу, що їх реалізує, та складні, як-от: MathMatrix, MathVector;
- 3) Mtrixes та Vectors — пакети, що включають в себе фабрики та класи матриць і векторів відповідно.

3 ВИСНОВОК

Під час лабораторної роботи опрацювали завдання з розробки паралельних алгоритмів множення матриць та дослідження їхньої ефективності. Результати записані до файлу, побудовано відповідні графіки. Було продемонстровано, що збільшення розмірності матриць суттєво впливає на час виконання алгоритму, а збільшення потоків зменшує час виконання операції. Хоча згідно із законом Амдала збільшення потоків впливає на час виконання операції нелінійно, а тому для досягнення більшої економії часу потрібно на порядок більше потоків для відчуття різниці.

4 ПОСИЛАННЯ

- 1) Amdahl's law. URL: https://en.wikipedia.org/wiki/Amdahl%27s_law

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду
(Найменування програми (документа))

Жорсткий диск

(Вид носія даних)

(Обсяг програми (документа), арк.)

Студента групи ІІІ-ІІІ курсу
Панченка С. В

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Enums/Coords.java
```

```
package org.LabMath.Enums;
```

```
public enum Coords {  
    Y,  
    X  
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Vectors/GeneralVector.java
```

```
package org.LabMath.Vectors;
```

```
import org.LabMath.Interfaces.MathVector;
```

```
import java.util.Arrays;
```

```
public class GeneralVector implements MathVector<GeneralVector> {  
    private static final String ERROR_LENGTHS_NOT_EQUAL = "Lengths of  
points arguments are not equal";  
    private double[] arguments;  
  
    public GeneralVector(int length) {  
        setLength(length);  
    }  
  
    public GeneralVector(GeneralVector other) {  
        setLength(other.getLength());  
        set(other);  
    }  
}
```

```

public int getLength() {
    if(arguments == null) {
        return 0;
    }
    return arguments.length;
}

```

```

public void setLength(int length) {
    var currentLength = getLength();

    if(currentLength==length) return;

    var minLength = Math.min(currentLength, length);
    var args = new double[length];
    for(var i = 0; i < minLength; ++i) {
        args[i] = getAt(i);
    }

    arguments = args;
}

```

@Override

```

public void set(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, other.getAt(i));
    }
}

```

@Override

```

public GeneralVector clone() {
    return new GeneralVector(this);
}

```

```
}
```

```
@Override
```

```
public String toString() {
    return Arrays.toString(arguments);
}
```

```
private void checkSizesEqual(GeneralVector other) {
    assert getLength() == other.getLength() : ERROR_LENGTHS_NOT_EQUAL;
}
```

```
@Override
```

```
public void add(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) + other.getAt(i));
    }
}
```

```
@Override
```

```
public void add(double value) {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) + value);
    }
}
```

```
@Override
```

```
public void sub(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) - other.getAt(i));
    }
}
```

```
}
```

```
@Override
```

```
public void sub(double value) {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) - value);
    }
}
```

```
@Override
```

```
public void mul(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) * other.getAt(i));
    }
}
```

```
@Override
```

```
public void mul(double value) {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) * value);
    }
}
```

```
@Override
```

```
public void div(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) / other.getAt(i));
    }
}
```


@Override

```
public void div(double value) {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) / value);
    }
}
```

@Override

```
public double getSize() {
    return Math.sqrt(getSizeSquared());
}
```

@Override

```
public double getSizeSquared() {
    double s = 0;
    for(var i = 0; i < getLength(); ++i) {
        s += Math.pow(getAt(i), 2);
    }
    return s;
}
```

@Override

```
public double getDotProduct(GeneralVector other) {
    checkSizesEqual(other);
    var prod = 0;
    for(var i = 0; i < getLength(); ++i) {
        prod += getAt(i) * other.getAt(i);
    }
    return prod;
}
```

@Override

```

public double getDistance(GeneralVector other) {
    checkSizesEqual(other);
    var dist = 0.0;
    for(var i = 0; i < getLength(); ++i) {
        dist += Math.pow(getAt(i) - other.getAt(i), 2);
    }
    dist = Math.sqrt(dist);
    return dist;
}

```

@Override

```

public GeneralVector getForwardVector() {
    var forwardVec = clone();
    var size = getSize();
    for(var i = 0; i < getLength(); ++i) {
        forwardVec.setAt(i, getAt(i) / size);
    }
    return forwardVec;
}

```

@Override

```

public double getAt(int index) {
    return arguments[index];
}

```

@Override

```

public void setAt(int index, double value) {
    arguments[index] = value;
}

```

@Override

```

public GeneralVector getOpposite() {

```

```

    var v = clone();
    v.toOpposite();
    return v;
}

```

```

@Override

```

```

public void toOpposite() {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, -getAt(i));
    }
}

```

```

}

```

```

// ./Lab2/Lab2/src/main/java/org/LabMath/Vectors/Vector2D.java

```

```

package org.LabMath.Vectors;

```

```

import org.LabMath.Enums.*;

```

```

import org.LabMath.Interfaces.MathVector;

```

```

public class Vector2D implements MathVector<Vector2D> {
    private final GeneralVector vec = new GeneralVector(2);

```

```

    public Vector2D() {}

```

```

    public Vector2D(double y, double x) {
        set(y, x);
    }

```

```

    public Vector2D(Vector2D other) {

```

```
        set(other);
    }

    public double getX() {
        return getAt(Coords.X.ordinal());
    }

    public double getY() {
        return getAt(Coords.Y.ordinal());
    }

    public void set(Vector2D other) {
        vec.set(other.vec);
    }

    public void set(double y, double x) {
        setX(x);
        setY(y);
    }

    public void setX(double value) {
        setAt(Coords.X.ordinal(), value);
    }

    public void setY(double value) {
        setAt(Coords.Y.ordinal(), value);
    }

    @Override
    public Vector2D clone() {
        return new Vector2D(getY(), getX());
    }
```

```
@Override  
public String toString() {  
    return vec.toString();  
}
```

```
@Override  
public void add(Vector2D other) {  
    vec.add(other.vec);  
}
```

```
@Override  
public void add(double value) {  
    vec.add(value);  
}
```

```
@Override  
public void sub(double value) {  
    vec.sub(value);  
}
```

```
@Override  
public void sub(Vector2D other) {  
    vec.sub(other.vec);  
}
```

```
@Override  
public void mul(Vector2D other) {  
    vec.mul(other.vec);  
}
```

```
@Override
```

```
public void mul(double value) {  
    vec.mul(value);  
}
```

```
@Override  
public void div(Vector2D other) {  
    vec.div(other.vec);  
}
```

```
@Override  
public void div(double value) {  
    vec.div(value);  
}
```

```
@Override  
public double getSize() {  
    return vec.getSize();  
}
```

```
@Override  
public double getSizeSquared() {  
    return vec.getSizeSquared();  
}
```

```
@Override  
public double getDotProduct(Vector2D other) {  
    return vec.getDotProduct(other.vec);  
}
```

```
@Override  
public double getDistance(Vector2D other) {  
    return vec.getDistance(other.vec);  
}
```

```
}
```

```
@Override
```

```
public Vector2D getForwardVector() {
    var forwardVec = clone();
    forwardVec.vec.set(forwardVec.vec.getForwardVector());
    return forwardVec;
}
```

```
@Override
```

```
public double getAt(int index) {
    return vec.getAt(index);
}
```

```
@Override
```

```
public void setAt(int index, double value) {
    vec.setAt(index, value);
}
```

```
@Override
```

```
public Vector2D getOpposite() {
    var v = clone();
    v.toOpposite();
    return v;
}
```

```
@Override
```

```
public void toOpposite() {
    vec.toOpposite();
}
```

```
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/MathVector.java
```

```
package org.LabMath.Interfaces;
```

```
import org.LabMath.Interfaces.General.*;
```

```
public interface MathVector<T> extends Cloneable, Divisible<T>, Multipliable<T>,
Addable<T>, Subtractable<T>,
```

```
    DoubleDivisible, DoubleMultipliable, DoubleAddable, DoubleSubtractable {
    double getSize();
    double getSizeSquared();
    double getDotProduct(T other);
    double getDistance(T other);
    T getForwardVector();
    double getAt(int index);
    void setAt(int index, double value);
    T getOpposite();
    void toOpposite();
    void set(T other);
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/MathMatrix.java
```

```
package org.LabMath.Interfaces;
```

```
import org.LabMath.Interfaces.General.*;
```

```
public interface MathMatrix<T> extends Cloneable, Addable<T>, Subtractable<T>,
Divisible<T>,
```

```
    GetMultipliable<T>, Settable<T>, DoubleSubtractable, DoubleMultipliable,
```



```
DoubleAddable, DoubleDivisible {
    int[] getDimensions();
    double getAt(int... indexes);
    void setAt(double value, int... indexes);
    int calcIndex(int... indexes);
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/General/DoubleDivisible.java
```

```
package org.LabMath.Interfaces.General;
```

```
public interface DoubleDivisible {
    void div(double other);
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/General/Divisible.java
```

```
package org.LabMath.Interfaces.General;
```

```
import jdk.jshell.spi.ExecutionControl;
```

```
public interface Divisible<T> {
    void div(T other) throws ExecutionControl.NotImplementedException;
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/General/Addable.java
```

```
package org.LabMath.Interfaces.General;
```

```
public interface Addable<T> {  
    void add(T other);  
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/General/Subtractable.java
```

```
package org.LabMath.Interfaces.General;
```

```
public interface Subtractable<T> {  
    void sub(T other);  
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/General/DoubleAddable.java
```

```
package org.LabMath.Interfaces.General;
```

```
public interface DoubleAddable {  
    void add(double other);  
}
```

```
//
```

```
./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/General/DoubleSubtractable.java
```

```
package org.LabMath.Interfaces.General;
```

```
public interface DoubleSubtractable {  
    void sub(double other);  
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/General/Settable.java
```

```
package org.LabMath.Interfaces.General;
```

```
public interface Settable<T> {  
    void set(T other);  
}
```

```
//
```

```
./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/General/DoubleMultipliable.java
```

```
package org.LabMath.Interfaces.General;
```

```
public interface DoubleMultipliable {  
    void mul(double other);  
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/General/Multipliable.java
```

```
package org.LabMath.Interfaces.General;
```

```
public interface Multipliable<T> {  
    void mul(T other);  
}
```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Interfaces/General/GetMultipliable.java
```

```
package org.LabMath.Interfaces.General;
```

```

import jdk.jshell.spi.ExecutionControl;

public interface GetMultipliable<T> {
    T getMul(T other) throws ExecutionControl.NotImplementedException;
}

// ./Lab2/Lab2/src/main/java/org/LabMath/Matrixes/Matrix2DFactory.java

package org.LabMath.Matrixes;

public class Matrix2DFactory {

    public Matrix2DFactory() {}

    public static void main(String[] args) {
        var factory = new Matrix2DFactory();
        var minVal = 0;
        var maxVal = 10;
        var rows = 5;
        var cols = 6;
        var one = factory.getRandom(rows, cols, minVal, maxVal);
        var two = factory.getRandom(cols, rows, minVal, maxVal);
        var result = one.getMul(two);
        System.out.println(result);
    }

    public Matrix2D getRandom(int rows, int cols, int minVal, int maxVal) {
        var res = new Matrix2D(rows, cols);
        for(var i = 0; i < rows; ++i) {
            for(var j = 0; j < cols; ++j) {

```

```

        res.setAt(Math.random() * (maxVal - minVal) + minVal, i, j);
    }
}
return res;
}
}

```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Matrixes/GeneralMatrix.java
```

```
package org.LabMath.Matrixes;
```

```
import org.LabMath.Interfaces.MathMatrix;
```

```
import org.LabMath.Vectors.GeneralVector;
```

```
import jdk.jshell.spi.ExecutionControl;
```

```
import java.util.Arrays;
```

```
public final class GeneralMatrix implements MathMatrix<GeneralMatrix> {
```

```
    private static final String ERROR_INDEXES = "Indexes are less than 0";
```

```
    private static final String ERROR_DIMENSIONS = "Matrix dimensions not  
equal";
```

```
    private static final String ERROR_DIMENSION_INDEXES = "Indexes length is  
not equal to amount of dimension";
```

```
    private final int[] dimensions;
```

```
    private final int total;
```

```
    private final GeneralVector mat;
```

```
    public GeneralMatrix(int... dimensions) {
```

```
        this.dimensions = dimensions.clone();
```

```
        var t = 1;
```

```
        for(var d : dimensions) t *= d;
```

```

    this.total = t;
    this.mat = new GeneralVector(this.total);
}

private String doDraw(int[] indexes, int dimension) {
    var res = new StringBuilder();
    res.append("{}");
    for(var i = 0; i < this.dimensions[dimension]; ++i) {
        indexes[dimension] = i;
        if(dimension == this.dimensions.length - 1) {
            res.append(this.mat.getAt(calcIndex(indexes)));
        } else {
            res.append(doDraw(indexes, dimension + 1));
        }
        res.append(this.dimensions[dimension] - 1 == i ? "" : ", ");
    }
    res.append("{}");
    return res.toString();
}

@Override
public String toString() {
    var indexes = new int[this.dimensions.length];
    return doDraw(indexes, 0);
}

private void checkDimensions(int[] dimensions) {
    if(!Arrays.equals(this.dimensions, dimensions)) {
        throw new IllegalArgumentException(ERROR_DIMENSIONS);
    }
}

```

```
private void checkIndexes(int[] indexes) {
    if(!Arrays.stream(indexes).allMatch(e -> e >= 0)) {
        throw new IllegalArgumentException(ERROR_INDEXES);
    }
}
```

@Override

```
public void add(GeneralMatrix other) {
    checkDimensions(other.dimensions);
    for(var i = 0; i < total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i) + other.mat.getAt(i));
    }
}
```

@Override

```
public void add(double value) {
    for(var i = 0; i < this.total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i) + value);
    }
}
```

@Override

```
public void div(double value) {
    for(var i = 0; i < this.total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i) / value);
    }
}
```

@Override

```
public void mul(double value) {
    for(var i = 0; i < this.total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i) * value);
    }
}
```

```

    }
}

```

@Override

```

public void sub(double value) {
    for(var i = 0; i < this.total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i) - value);
    }
}

```

@Override

```

        public GeneralMatrix getMul(GeneralMatrix other) throws
ExecutionControl.NotImplementedException {
    throw new ExecutionControl.NotImplementedException("");
}

```

@Override

```

public void set(GeneralMatrix other) {
    checkDimensions(other.dimensions);
    for(var i = 0; i < this.total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i));
    }
}

```

@Override

```

public void sub(GeneralMatrix other) {
    checkDimensions(other.dimensions);
    for(var i = 0; i < total; ++i) {
        this.mat.setAt(i, other.mat.getAt(i) - other.mat.getAt(i));
    }
}

```


@Override

```
public int[] getDimensions() {
    return dimensions.clone();
}
```

@Override

```
public double getAt(int... indexes) {
    checkIndexes(indexes);
    return this.mat.getAt(this.calcIndex(indexes));
}
```

@Override

```
public void setAt(double value, int... indexes) {
    checkIndexes(indexes);
    var index = this.calcIndex(indexes);
    this.mat.setAt(index, value);
}
```

@Override

public void div(GeneralMatrix other) throws

```
ExecutionControl.NotImplementedException {
    throw new ExecutionControl.NotImplementedException("");
}
```

@Override

```
public int calcIndex(int... indexes) {
    if(indexes.length != dimensions.length) {
        throw new IllegalArgumentException(ERROR_DIMENSION_INDEXES);
    }
    var index = 0;
    var mult = 1;
    for(var i : dimensions) mult *= i;
```

```

        for(var i = 0; i < indexes.length; ++i) {
            mult /= dimensions[i];
            index += indexes[i] * mult;
        }
        return index;
    }
}

```

```
// ./Lab2/Lab2/src/main/java/org/LabMath/Matrixes/Matrix2D.java
```

```
package org.LabMath.Matrixes;
```

```
import org.LabMath.Interfaces.MathMatrix;
```

```
import jdk.jshell.spi.ExecutionControl;
```

```
public class Matrix2D implements MathMatrix<Matrix2D> {
```

```
    private static final String ERROR_MULTIPLICATION = "Rows and columns are  
not equal";
```

```
    private static final String ERROR_INDEXES = "Indexes are less than 0";
```

```
    private final int rows;
```

```
    private final int cols;
```

```
    private final GeneralMatrix mat;
```

```
    public static void main(String[] args) {}
```

```
    @Override
```

```
    public String toString() {
```

```
        return mat.toString();
```

```
    }
```

```
    public Matrix2D(int rows, int cols) {
```

```

    mat = new GeneralMatrix(rows, cols);
    this.rows = rows;
    this.cols = cols;
}

```

```

public int getRows() {
    return this.rows;
}

```

```

public int getCols() {
    return this.cols;
}

```

```

@Override
public void add(Matrix2D other) {
    this.mat.add(other.mat);
}

```

```

@Override
public void div(Matrix2D other) throws

```

```

ExecutionControl.NotImplementedException {
    throw new ExecutionControl.NotImplementedException("");
}

```

```

@Override
public void add(double value) {
    this.mat.add(value);
}

```

```

@Override
public void div(double value) {
    this.mat.div(value);
}

```

```
}
```

```
@Override
```

```
public void mul(double value) {
    this.mat.mul(value);
}
```

```
@Override
```

```
public void sub(double value) {
    this.mat.sub(value);
}
```

```
@Override
```

```
public Matrix2D getMul(Matrix2D other) {
    var cols = getCols();

    assert cols == other.getRows() : ERROR_MULTIPLICATION;

    var result = new Matrix2D(rows, cols);

    for(var i = 0; i < getRows(); ++i) {
        for(var j = 0; j < other.getCols(); ++j) {
            var value = 0;
            for(var k = 0; k < cols; ++k) {
                value += this.mat.getAt(i, k) * other.mat.getAt(k, j);
            }
            result.setAt(value, i, j);
        }
    }

    return result;
}
```

@Override

```
public void set(Matrix2D other) {  
    this.mat.set(other.mat);  
}
```

@Override

```
public void sub(Matrix2D other) {  
    this.mat.sub(other.mat);  
}
```

@Override

```
public int[] getDimensions() {  
    return this.mat.getDimensions();  
}
```

@Override

```
public double getAt(int... indexes) {  
    if(indexes.length != 2) {  
        throw new IllegalArgumentException(ERROR_INDEXES);  
    }  
    return this.mat.getAt(indexes);  
}
```

@Override

```
public void setAt(double value, int... indexes) {  
    if(indexes.length != 2) {  
        throw new IllegalArgumentException(ERROR_INDEXES);  
    }  
    this.mat.setAt(value, indexes);  
}
```

@Override

```
public int calcIndex(int... indexes) {
    return this.mat.calcIndex(indexes);
}
```

```
public boolean isSquare() {
    return this.rows == this.cols;
}
}
```

//

./Lab2/Lab2/src/main/java/org/MultiplicationAlgorithms/BlockStripedAlgorithm.java

```
package org.MultiplicationAlgorithms;
```

```
import org.LabMath.Matrixes.Matrix2D;
import org.LabMath.Matrixes.Matrix2DFactory;
```

```
public class BlockStripedAlgorithm extends GeneralAlgorithm {
```

```
    public BlockStripedAlgorithm() {}
```

```
    public BlockStripedAlgorithm(int threadsNum, Matrix2D first, Matrix2D second)
    {
        super(threadsNum, first, second);
    }
```

```
    public static void main(String[] args) {
        var matrixFactory = new Matrix2DFactory();
        var rows = 10;
        var cols = 10;
```

```

var minVal = 0;
var maxVal = 10;
var threadsNum = 5;
var first = matrixFactory.getRandom(rows, cols, minVal, maxVal);
var second = matrixFactory.getRandom(rows, cols, minVal, maxVal);
var algorithm = new BlockStripedAlgorithm(threadsNum, first, second);
var result = algorithm.solve();
System.out.println("First:\t" + first);
System.out.println("Second:\t" + second);
System.out.println("Result:\t" + result);
}

public Matrix2D solve() {
    var firstRows = first.getRows();
    var firstCols = first.getCols();
    var secondRows = second.getRows();
    var secondCols = second.getCols();

    if(firstCols != secondRows) {
        throw new IllegalArgumentException(ERROR_MULTIPLICATION);
    }

    var result = new Matrix2D(firstRows, secondCols);
    var isRowsLess = firstRows < threads.length;
    var totalThreads = isRowsLess ? firstRows : threads.length;
    var step = isRowsLess ? 1 : threads.length;

    for(var i = 0; i < totalThreads; ++i) {
        threads[i] = new BlockStripedThread(i, step, first, second, result);
    }

    for(var i = 0; i < totalThreads; ++i) {

```

```

        threads[i].start();
    }

    for(var i = 0; i < totalThreads; ++i) {
        try {
            threads[i].join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    return result;
}
}

// ./Lab2/Lab2/src/main/java/org/MultiplicationAlgorithms/FoxAlgorithmThread.java

package org.MultiplicationAlgorithms;

import org.LabMath.Matrixes.Matrix2D;

public class FoxAlgorithmThread extends GeneralAlgorithmThread {
    private final int start;
    private final int blockSize;

    public FoxAlgorithmThread(int index, int blockSize, Matrix2D first, Matrix2D
second, Matrix2D result) {
        super(first, second, result);
        this.start = index * blockSize;
        this.blockSize = blockSize;
    }
}

```


@Override

```
public void run() {
    var curRow = start;
    var curCol = start;
    var rows = first.getRows();
    var steps = rows / blockSize;
    for(var i = 0; i < steps; ++i) {
        for(var j = 0; j < steps; ++j) {
            mulMatrices(curCol, curRow, j * blockSize);
        }
        curRow += blockSize;
        curRow %= rows;
        curCol += blockSize;
        curCol %= rows;
    }
}

public void mulMatrices(int firstCol, int secondRow, int secondCol) {
    for(var i = start; i < start + blockSize; ++i) {
        for(var j = secondCol; j < secondCol + blockSize; ++j) {
            var value = 0.0;
            for(var k = 0; k < blockSize; ++k) {
                value += first.getAt(i, firstCol + k) * second.getAt(secondRow + k, j);
            }
            result.setAt(result.getAt(i, j) + value, i, j);
        }
    }
}
```

```
// ./Lab2/Lab2/src/main/java/org/MultiplicationAlgorithms/FoxAlgorithm.java
```

```
package org.MultiplicationAlgorithms;
```

```
import org.LabMath.Matrixes.Matrix2D;
```

```
import org.LabMath.Matrixes.Matrix2DFactory;
```

```
public class FoxAlgorithm extends GeneralAlgorithm {
```

```
    private static final String ERROR_PROCS_NUM = "Number of processes must be  
a square number";
```

```
    private static final String ERROR_SQUARE_MATRIX = "Rows and columns are  
not equal";
```

```
    private static final String ERROR_BLOCK_SIZE = "Matrix size must be divisible  
by number of threads";
```

```
    public FoxAlgorithm() {}
```

```
    public FoxAlgorithm(int threadsNum, Matrix2D first, Matrix2D second) {  
        super(threadsNum, first, second);  
    }
```

```
    public static void main(String[] args) {
```

```
        var matrixFactory = new Matrix2DFactory();
```

```
        var rows = 3;
```

```
        var cols = 3;
```

```
        var minVal = 0;
```

```
        var maxVal = 10;
```

```
        var threadsNum = 3;
```

```
        var first = matrixFactory.getRandom(rows, cols, minVal, maxVal);
```

```
        var second = matrixFactory.getRandom(rows, cols, minVal, maxVal);
```

```

var algorithm = new FoxAlgorithm(threadsNum, first, second);
var striped = new BlockStripedAlgorithm(threadsNum, first, second);
var result = algorithm.solve();
var stripedResult = striped.solve();
System.out.println("First:\t" + first);
System.out.println("Second:\t" + second);
System.out.println("Fox:\t" + result);
System.out.println("Striped:\t" + stripedResult);
}

private void checkIfSquare(Matrix2D matrix) {
    if(!matrix.isSquare()) {
        throw new IllegalArgumentException(ERROR_SQUARE_MATRIX);
    }
}

@Override
public void setFirst(Matrix2D first) {
    checkIfSquare(first);
    super.setFirst(first);
}

@Override
public void setSecond(Matrix2D second) {
    checkIfSquare(second);
    super.setSecond(second);
}

@Override
public Matrix2D solve() {
    var rows = first.getRows();

```

```
var blockSize = rows / threads.length;
```

```
if(rows % threads.length != 0) {
```

```
    throw new IllegalArgumentException(ERROR_BLOCK_SIZE);
```

```
}
```

```
var matrices = new Matrix2D[threads.length];
```

```
for(var i = 0; i < threads.length; ++i) {
```

```
    matrices[i] = new Matrix2D(rows, rows);
```

```
    threads[i] = new FoxAlgorithmThread(i, blockSize, first, second, matrices[i]);
```

```
    threads[i].start();
```

```
}
```

```
for(var t : threads) {
```

```
    try {
```

```
        t.join();
```

```
    } catch (InterruptedException e) {
```

```
        throw new RuntimeException(e);
```

```
    }
```

```
}
```

```
if(threads.length == 1) {
```

```
    return matrices[0];
```

```
}
```

```
for(var i = 1; i < matrices.length; ++i) {
```

```
    matrices[0].add(matrices[i]);
```

```
}
```

```
return matrices[0];
```

```
}
```

```
}
```

```
//
./Lab2/Lab2/src/main/java/org/MultiplicationAlgorithms/GeneralAlgorithmThread.java

package org.MultiplicationAlgorithms;

import org.LabMath.Matrixes.Matrix2D;

public abstract class GeneralAlgorithmThread extends Thread {
    protected Matrix2D first;
    protected Matrix2D second;
    protected Matrix2D result;

    public GeneralAlgorithmThread(Matrix2D first, Matrix2D second, Matrix2D
result) {
        this.first = first;
        this.second = second;
        this.result = result;
    }

    @Override
    public abstract void run();
}

// ./Lab2/Lab2/src/main/java/org/MultiplicationAlgorithms/BlockStripedThread.java

package org.MultiplicationAlgorithms;

import org.LabMath.Matrixes.Matrix2D;
```

```

public class BlockStripedThread extends GeneralAlgorithmThread {
    private final int step;
    private final int firstRow;

    public BlockStripedThread(int firstRow, int step, Matrix2D first, Matrix2D
second, Matrix2D result) {
        super(first, second, result);
        this.firstRow = firstRow;
        this.step = step;
    }

    @Override
    public void run() {
        var firstRows = first.getRows();
        var firstCols = first.getCols();
        var secondCols = second.getCols();
        var curRow = firstRow;
        while(curRow < firstRows) {
            for(var j = 0; j < secondCols; ++j) {
                var value = 0.0;
                for(var k = 0; k < firstCols; ++k) {
                    value += first.getAt(curRow, k) * second.getAt(k, j);
                }
                result.setAt(value, curRow, j);
            }
            curRow += step;
        }
    }
}

```

```
// ./Lab2/Lab2/src/main/java/org/MultiplicationAlgorithms/GeneralAlgorithm.java
```

```
package org.MultiplicationAlgorithms;
```

```
import org.LabMath.Matrixes.Matrix2D;
```

```
public abstract class GeneralAlgorithm {
```

```
    protected static final String ERROR_MULTIPLICATION = "Rows and columns  
are not equal";
```

```
    protected static final String ERROR_NUM_OF_THREADS = "Number of threads  
must be positive";
```

```
    protected Thread[] threads;
```

```
    protected Matrix2D first;
```

```
    protected Matrix2D second;
```

```
    public GeneralAlgorithm() {}
```

```
    GeneralAlgorithm(int threadsNum, Matrix2D first, Matrix2D second) {
```

```
        setThreadsNum(threadsNum);
```

```
        setFirst(first);
```

```
        setSecond(second);
```

```
    }
```

```
    public void setThreadsNum(int threadsNum) {
```

```
        if(threadsNum <= 0) {
```

```
            throw new IllegalArgumentException(ERROR_NUM_OF_THREADS);
```

```
        }
```

```
        if(this.threads != null && this.threads.length == threadsNum) return;
```

```
        this.threads = new Thread[threadsNum];
```

```
    }
```

```
public void setFirst(Matrix2D first) {  
    this.first = first;  
}
```

```
public void setSecond(Matrix2D second) {  
    this.second = second;  
}
```

```
public abstract Matrix2D solve();  
}
```

```
// ./Lab2/Lab2/src/test/java/test/MatrixTester.java
```

```
package test;
```

```
import com.github.sh0nk.matplotlib4j.*;  
import org.LabMath.Matrixes.Matrix2DFactory;  
import org.MultiplicationAlgorithms.BlockStripedAlgorithm;  
import org.MultiplicationAlgorithms.FoxAlgorithm;  
import org.MultiplicationAlgorithms.GeneralAlgorithm;
```

```
import java.awt.*;  
import java.io.*;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Collections;
```

```
public class MatrixTester {  
    private static final int minVal = -10;  
    private static final int maxVal = 10;  
    private static final String FILE_NAME = "results.csv";
```



```

private static final String DELIMETER = "\t";
private static final String LEGEND_POSITION = "upper left";
private static final String X_LABEL = "Matrix size";
private static final String Y_LABEL = "Milliseconds";
private static final String SPEEDUP_Y_LABEL = "Speedup";
private static final String PLOT_FILE = "plot.png";
private static final String SPEEDUP_PLOT_FILE = "speedup_plot.png";

```

```

private static class AlgorithmResult {

```

```

    public final long milliseconds;
    public final int threadsNum;
    public final int size;
    public final String name;
    public final double speedup;

```

```

        public AlgorithmResult(long time, int threadsNum, int size, double speedup,
String name) {
            this.milliseconds = time;
            this.threadsNum = threadsNum;
            this.size = size;
            this.name = name;
            this.speedup = speedup;
        }

```

```

    @Override

```

```

    public String toString() {
        return String.format("%s\t%d\t%d\t%d\t%f", name, threadsNum, size,
milliseconds, speedup);
    }
}

```

```

    public static void main(String[] args) throws PythonExecutionException,
IOException {
        var tester = new MatrixTester();
        //      var threadsNums = new int[] {1, 4, 10};
        //      var matrixSizes = new int[] {100, 200, 300, 400, 500, 600, 700};
        //      var algorithms = new GeneralAlgorithm[] {new FoxAlgorithm(), new
BlockStripedAlgorithm()};
        //      tester.testAlgorithm(threadsNums, matrixSizes, algorithms);

        tester.plotStatistic();
        tester.plotSpeedup();
    }

```

```

    public void plotStatistic() throws PythonExecutionException, IOException {
        Plot plt = Plot.create();
        var results = readStatistic();
        var algorithms = results.stream().map(r -> r.name).distinct().toList();
        var threadNums = results.stream().map(r -> r.threadsNum).distinct().toList();
        for(var a : algorithms) {
            var filtered = results.stream().filter(r -> r.name.equals(a)).toList();
            for(var threadNum : threadNums) {
                var filteredByThreadNum = filtered.stream().filter(r -> r.threadsNum ==
threadNum).toList();
                var x = filteredByThreadNum.stream().map(r -> r.size).toList();
                var y = filteredByThreadNum.stream().map(r -> r.milliseconds).toList();
                plt.plot().add(x, y).label(a + " " + threadNum);
            }
        }
        plt.legend().loc(LEGEND_POSITION);
        plt.xlabel(X_LABEL);
        plt.ylabel(Y_LABEL);
        plt.savefig(PLOT_FILE);
    }

```

```
plt.show();
}
```

```
private void plotSpeedup() throws IOException, PythonExecutionException {
    Plot plt = Plot.create();
    var results = readStatistic();
    var algorithms = results.stream().map(r -> r.name).distinct().toList();
    var threadNums = results.stream().map(r -> r.threadsNum).distinct().toList();
    for(var a : algorithms) {
        var filtered = results.stream().filter(r -> r.name.equals(a)).toList();
        for(var threadNum : threadNums) {
            var filteredByThreadNum = filtered.stream().filter(r -> r.threadsNum ==
threadNum).toList();
            var x = filteredByThreadNum.stream().map(r -> r.size).toList();
            var y = filteredByThreadNum.stream().map(r -> r.speedup).toList();
            plt.plot().add(x, y).label(a + " " + threadNum);
        }
    }
    plt.legend().loc(LEGEND_POSITION);
    plt.xlabel(X_LABEL);
    plt.ylabel(SPEEDUP_Y_LABEL);
    plt.savefig(SPEEDUP_PLOT_FILE);
    plt.show();
}
```

```
private ArrayList<AlgorithmResult> readStatistic() throws IOException {
    var line = "";
    BufferedReader br = new BufferedReader(new FileReader(FILE_NAME));
    var results = new ArrayList<AlgorithmResult>();
    while ((line = br.readLine()) != null) {
```

```

String[] row = line.split(DELIMITER);
results.add(
    new AlgorithmResult(
        Long.parseLong(row[3]),
        Integer.parseInt(row[1]),
        Integer.parseInt(row[2]),
        Double.parseDouble(row[4]),
        row[0]
    )
);
}
return results;
}

```

```

public void testAlgorithm(int[] threadNums, int[] matrixSizes, GeneralAlgorithm[]
algorithms) throws IOException {
    var file = new File( FILE_NAME);
    var results = new FileOutputStream(file);
    var matrixFactory = new Matrix2DFactory();
    for(var algorithm : algorithms) {
        var algName = algorithm.getClass().getSimpleName();
        for(var size : matrixSizes) {
            long threadTimeOne = 0;
            for(var threadsNum : threadNums) {

                var startTime = System.currentTimeMillis();

                var first = matrixFactory.getRandom(size, size, minVal, maxVal);
                var second = matrixFactory.getRandom(size, size, minVal, maxVal);

                algorithm.setThreadsNum(threadsNum);
                algorithm.setFirst(first);
            }
        }
    }
}

```

```
algorithm.setSecond(second);
```

```
algorithm.solve();
```

```
var endTime = System.currentTimeMillis();
```

```
var duration = endTime - startTime;
```

```
if(threadsNum==1) threadTimeOne = duration;
```

```
var result = new AlgorithmResult(  
    duration, threadsNum, size,  
    threadTimeOne / (double) duration , algName).toString();
```

```
System.out.println(result);
```

```
results.write((result + "\n").getBytes());
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```